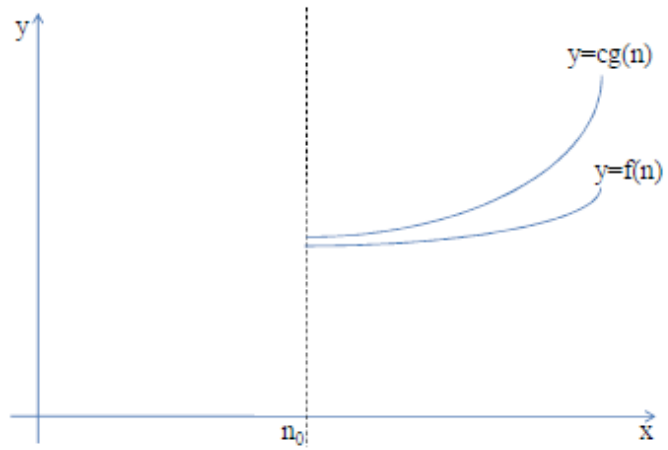
Order of growth

CS330 – Lecture 02 Asymptotic Notation

Reference: Section2. 3, Introduction to the Design and Analysis of Algorithms by Levitin

Big O



Beyond $n = n_0$, $cg(n) \geq f(n)$

$c \cdot g(n)$

$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

$\Downarrow$

$f(n) \in O(g(n)) \Rightarrow O(g(n))$ family.

$f(n) = n+5$ $\quad$ $g(n) = n$

$f(n) \in O(g(n))$?

$(n+5) \in O(n)$ true.

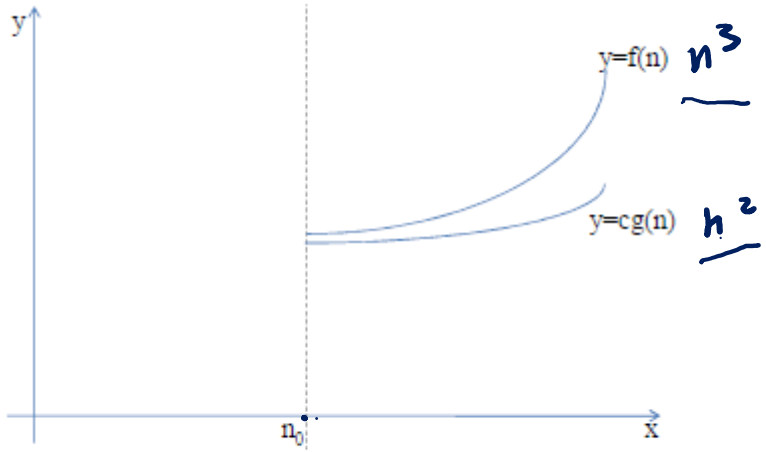
$n+5 \leq c \times n$. · Let $c=2$

$n_0 = 5$

$n+5 \leq 2n$ for all $n \geq 5$ holds.

Big Ω

Definition



Beyond $n = n_0$, $cg(n) \geq f(n)$

$f(n) \geq C \cdot g(n)$ for all $n \geq n_0$

$\Downarrow$

$f(n) \in \Omega(g(n))$

Example

$f(n) = n^3$ $\quad$ $g(n) = n^2$

$n^3 \geq C \cdot n^2$ for $n \geq n_0$, let $C=1$, $n_0 = 0$

$f(n) \in \Omega(g(n))$

$n^3 \in \Omega(n^2)$

$n^3 \geq n^2$. for $n \geq 0$

$\frac{n(n+1)}{2} \in \Omega(n)$ true.

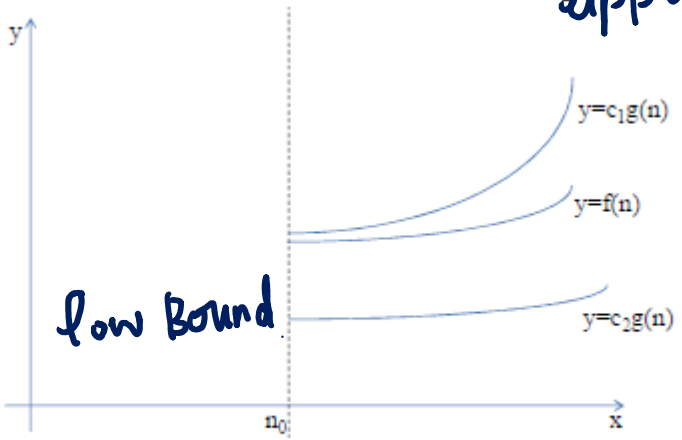
$f(n) = \frac{n(n+1)}{2}$ $\quad$ $g(n) = n$

$\frac{1}{2}n^2 + \frac{n}{2} \geq n$ $\quad$ for all $n \geq 0$. $C = 1$

$c_2 g(n) \le f(n) \le c_1 g(n)$ for all $n \ge n_0$

upper bound.

$f(n) \in \Theta(g(n))$



$f(n)$ remains firmly between $c_1 g(n)$ and $c_2 g(n)$ after $n = n_0$

Big Θ

$f(n) = \frac{1}{2} n(n-1)$

$g(n) = n^2$

$f(n) \le c_1 g(n)$

$\frac{1}{2} n(n-1) = \frac{1}{2} n^2 - \frac{1}{2} n \le \frac{1}{2} n^2$ for all $n \ge 0$

↳ Let $c_1 = \frac{1}{2}$ □

$c_2 g(n) \le f(n)$

$\frac{1}{2} n(n-1) = \frac{1}{2} n^2 - \frac{1}{2} n \ge \frac{1}{2} n^2 - \frac{n}{2} \cdot \frac{n}{2}$ for all $n \ge 2$

$= \frac{1}{2} n^2 - \frac{n^2}{4} = \frac{1}{4} n^2$

↳ Let $c_2 = \frac{1}{4}$ □

⇓

$\frac{1}{2} n(n-1) \in \Theta(n^2)$

$\frac{1}{2} n(n-1) \in \Theta(n^2)$ for all $n \ge 2$.

$(c_1 = \frac{1}{2}, c_2 = \frac{1}{4})$

$f(n) = \frac{n(n+1)}{2}$

count the number of times the basic operation is executed on the size of the input.

$f(n)$ is polynomial

⇓

order of growth

$(0, \Omega, \Theta)$

Analyzing Non-recursive Algorithm

Example 1

$n-1 \in \Theta(n)$

ALGORITHM *MaxElement*$(A[0..n-1])$ n.

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

Basic Operation

$f(n) = \sum_{i=1}^{n-1} 1 = n-1$

$f(n) \leq c \cdot g(n)$. $c_1 = 1$

$n-1 \leq n$ for all $n \geq 0$.

$n-1 \geq \frac{1}{2}n$ for all $n \geq 2$

$c = \frac{1}{2}$.

$\in O(n)$ ✓

$\in \Omega(n)$

$f(n) \in \Theta(n)$ for all $n \geq 2$.

Example 2

[1 1 2 3 4] → Θ(1) Best case

worst case

input size : n.

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
// and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[i] = A[j]$ **return false**
return true

Basic operator.

i		# time
0 :	1.. n-1	n-1
1 :	2.. n-1	n-2
2 :	3.. n-1	n-3
⋮		⋮
n-1 :	n-2..n-1	2
n-2 :	n-1..n-1	1
		0

$f(n) = 1+2+3+\cdots+(n-1)$

$1+2+\cdots+n$

$\text{sum} = \frac{n(1+n)}{2}$

$= \frac{(n-1)(1+n-1)}{2} = \frac{(n-1)\cdot n}{2} \in \Theta(n^2)$

$= \frac{n^2-n}{2} = \frac{1}{2}n^2 - \frac{1}{n}$

Example 3

size of input. n.

value of n.

$f(9) = \lfloor \log_2 9 \rfloor = 3$

$\Rightarrow f(n) = \log_2 n$.

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

$count \leftarrow 1$

while $n > 1$ **do**

$count \leftarrow count + 1$

$n \leftarrow \lfloor n/2 \rfloor$ ← basic operation.

return $count$

$f(8) = \log_2 8 = 3$.

$f(4) = \log_2 4 = 2$.

$\log_2 n \in \Theta(\log n)$

$n = 9$

# iter 1	$9 > 1$	1 time	$n = \lfloor 9/2 \rfloor = 4$
# iter 2	$4 > 1$	1 time	$n = \lfloor 4/2 \rfloor = 2$
# iter 3	$2 > 1$	1 time	$n = 2/2 = 1$
# iter 4	$1 > 1$	false	

} 3.

Comparing orders of growth

① ② Analyze the efficiency of recursive algorithm

The time efficiencies of a large number of algorithms fall into only a few classes. These classes are listed in Table 2.2 in increasing order of their orders of growth, along with their names and a few comments.

TABLE 2.2 Basic asymptotic efficiency classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 4.4). Note that a logarithmic algorithm cannot take into account all its input or even a fixed fraction of it: any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	*linearithmic*	Many divide-and-conquer algorithms (see Chapter 5), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on $n \times n$ matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.

Using limits to compare the order of growth

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n), \\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n), \\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n). \end{cases}$$

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n),\\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n),\\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n).\end{cases}$$

$\frac{1}{2}n(n-1)$ and n^2

$t(n)=\frac{1}{2}n(n-1)$ $\quad g(n)=n^2$

$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\lim_{n\to\infty}\frac{\frac{1}{2}n(n-1)}{n^2}=\frac{1}{2}\lim_{n\to\infty}\frac{n^2-n}{n^2}$

$t(n)=\frac{1}{2}(n^2-n)$

$=\frac{1}{2}n^2-\frac{1}{2}n$

$=\frac{1}{2}\lim_{n\to\infty}\left(1-\frac{n}{n^2}\right)=\frac{1}{2}\lim_{n\to\infty}\left(1-\frac{1}{n}\right)=\frac{1}{2}$

$\in\Theta(n^2)$

or $\in O(n^2)$

$\frac{1}{2}n(n-1)$ has the same order of growth as n^2 (up to a positive constant)

$\Rightarrow \frac{1}{2}n(n-1)\in\Theta(n^2)$

$\log_2 n < \sqrt{n}$ ✓

$\sqrt{n} < \log_2 n$ ×

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n), \\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n), \\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n). \end{cases}$$

$\log_2 n$ and $\sqrt{n}$.

$t(n) = \log_2 n \qquad g(n) = n^{\frac{1}{2}}$

$$\lim_{n\to\infty} \frac{\log_2 n}{n^{\frac{1}{2}}} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(n^{\frac{1}{2}})'} = \lim_{n\to\infty} \frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2}\cdot\frac{1}{n^{\frac{1}{2}}}}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n} = 2\log_2 e \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$$

$\log_2 n$ has a small order of growth than $g(n)$

$n!$ and 2^n

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

when n gets larger.

$f(n) = n!$ $\quad g(n) = 2^n$

$$\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n}\,(\frac{n}{e})^n}{2^n} =$$

$$\lim_{n\to\infty} \sqrt{2\pi n}\,(\frac{n}{2e})^n \nearrow = \infty$$

$n!$ has a larg

Factorial n has a larger order of growth than 2^n